Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

SCHEDULING CLUSTER TOOLS USING FILTERED BEAM SEARCH AND RECIPE COMPARISON

Simon Oechsner

Department of Computer Science
University of Würzburg
D-97074 Würzburg, GERMANY

Oliver Rose

Department of Applied Computer Science
Technical University of Dresden
D-01062 Dresden, GERMANY

ABSTRACT

Cluster tools have gained a lot of importance in today's semiconductor manufacturing. A cluster tool basically consists of several processing chambers in a mainframe, several load locks to insert wafer lots and a robot arm to move them. This means that these tools are able to work on more than one lot at the same time. Since the lot combination processed together can have an influence on the cycle times of these lots, scheduling is needed to ensure that the overall cycle times are kept low. In a previous work, we presented a method based on filtered beam search using slowdown factors as evaluation methods. Here, we will present another evaluation method based on recipe comparison that produces even better results. We will also show results of a beam width parameter study.

1 INTRODUCTION

In semiconductor manufacturing, cluster tools have become more and more important because of their potential for cost reduction. By performing several process steps in one machine, the clean-room space needed is minimized as well as the danger of pollution. Moreover, cluster tools offer speed advantages by pipelining. The basic structure of a cluster tool is shown in Figure 1.

The tool consists of several processing chambers which are connected to a mainframe. The chambers are the machines that perform the actual wafer processing. There are different chambers for different tasks, like alignment, heating, etching etc. Each processing step takes time. During that time the chamber is blocked, because there is no waiting space inside the cluster tool. Load locks are the interfaces to the outside of the tool. Each load lock can be loaded with one lot. Then the load lock adjusts the air pressure and cleanliness to a level suitable for the interior of the cluster tool. The handler is a robot that moves the wafers around. It takes a wafer out of its lot cassette or chamber and puts it in the chamber that is next in the wafers recipe, i.e., list of processing steps.

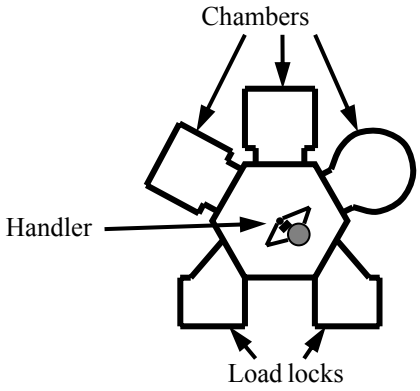

Figure 1: Sketch of a Typical Cluster Tool

Since there is more than one load lock in typical cluster tools, more than one lot can be processed at the same time in the tool. Of course, the wafers of these lots compete for the resources like handler and chambers and therefore influence each others cycle time.

The scheduling of the wafer movements, the so-called internal scheduling, is normally done by the cluster tool. However, the use of the chambers strongly depends on the sequence in which a given set of lots enters the tool. For example, if two lots of the same type are in the tool at the same time, the wafers from these lots need to be processed in the same chambers. Therefore, waiting times for at least some of the wafers can occur. However, if lots that use different resources are combined in the cluster tool, the overall processing time may be significantly reduced.

The choice of the lot order influences the makespan of that lot set, so the external scheduling tries to find a lot sequence that minimizes the makespan for a given lot set (Figure 2).

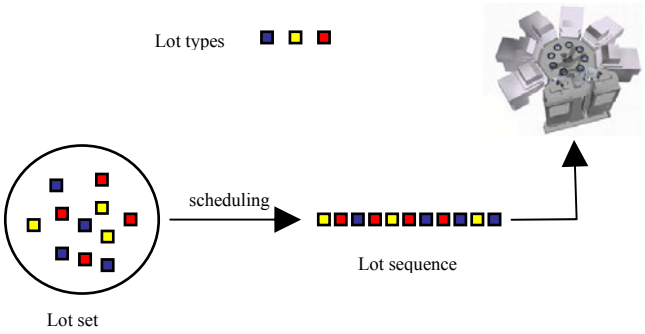

Figure 2: The Scheduling Process for Our Problem

2 RELATED WORK

In (Dümmler 2004), M. Dümmler tried to optimize small lot sequences by using a genetic algorithm. While being successful, the general purpose genetic algorithm lacks performance when used on specific problems that could be solved faster when using knowledge about the system. More work was done on the internal scheduling (Bohr 1999), as well as on the approximation of cycle times in the cluster tool (Niedermayer and Rose 2003) . In an earlier work (Oechsner and Rose 2005), we described the filtered beam search algorithm with evaluation methods based on slowdown factor comparison.

3 EXTERNAL SCHEDULING WITH FILTERED BEAM SEARCH

In this section, we will discuss the basic filtered beam search algorithm that we used, as well as the actual evaluation methods we implemented.

3.1 Schedule Trees

To introduce the methodology of our search algorithm, we first have to model the scheduling process by means of a tree structure. Each node of the tree represents a (partial) schedule. The root is the sequence with no lot scheduled on a distinct position (i.e., (*,*,*,*), with * being a placeholder for one slot in the sequence). Each child is a partial schedule with one lot scheduled on the first position of the sequence and for each child this lot is different (e.g., (1,*,*,*), (2,*,*,*), etc.). On each consecutive level, one more lot is put into the sequence, until full sequences/schedules are reached (Figure 3). These are the leaves of the tree. This means that on level i, one of the lots still to be scheduled is selected for position i of the lot sequence. With a finite set of different lots, the number of children per node decreases by one on each level, since fewer lots remain to be scheduled. If we have a set of lots that consists of a few lot types only with several lots per type, the initial number of children per node is of course smaller (the number of types/recipes), but it decreases only if all lots of one type are already scheduled.

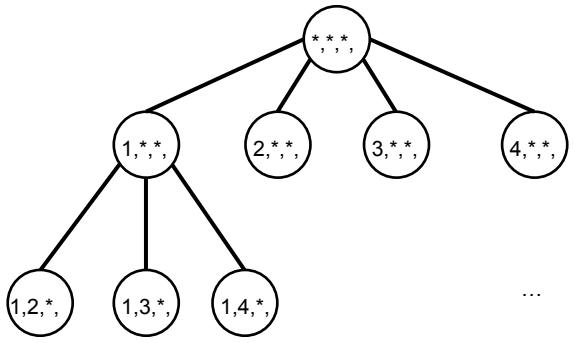

Figure 3: A Part of a Schedule Tree with 4 Different Lots

3.2 The Filtered Beam Search Algorithm

Filtered beam search (FBS) is a variant of the branch and bound algorithm (Pinedo 2001). Consider the scheduling process as a tree like in 3.1. For a large set of jobs, this tree becomes very big because of the large number of children of each node on higher levels. Branch and bound now aims to eliminate some of the children of a node by evaluating each node and comparing the resulting value with a provable lower bound. If the value is larger, the node and all of its children are discarded. Thus fewer nodes have to be considered on the next level. However, usually many nodes remain that have to be evaluated. While a branch and bound algorithm produces optimal solutions, it is very time-consuming if used on NP-hard problems. Since scheduling falls into this category, branch and bound is not the best option when results (not necessarily optimal) need to be obtained fast.

The aim of filtered beam search is to limit the number of nodes that have to be evaluated on each level of the search tree. With each step, a certain set of nodes is selected based on an evaluation function, the rest is discarded. Only the surviving nodes are expanded, keeping the size of the tree small. The number of these nodes is called the *beam width* of the search. When the search tree has been expanded fully, this means that on each level except the first, there are at most *beam width* nodes. This results in the same number of lot sequences that have to be simulated. Thus, the algorithm is quicker than branch and bound, but it can not guarantee optimality anymore. However, if the selection process for the nodes is good, a near optimal or even optimal solution can still be obtained. On the other hand, if the evaluation is too complex, the speed advantage might be lost.

To achieve acceptable speed while still gaining a good result, a filter is used. The filter is used on all nodes of a level. It uses a quick evaluation method to select a number of candidate nodes which are then evaluated thoroughly as

described above. The number of these nodes is called the *filter width* of the algorithm. It is clear that the beam width can not be larger than the filter width. Of course, if a fast but still thorough evaluation method can be found, the filter step can be omitted. In this work, we aim to develop fast methods that still produce good results.

3.3 Evaluation Functions

As the framework of our algorithm is given, we now need to find adequate evaluation functions used to prune the search tree as described above. These should be appropriate for the problem given, so we have to analyze the mode of operation of a cluster tool.

In reality, most cluster tools have one or two load locks. Since cluster tools with one load lock are quite simple to analyze from a schedulers point of view we concentrate on cluster tools with two load locks.

To evaluate a certain node of the search tree, we have to decide how well it fits to the end of the partial schedule that was already determined. Because we have only two load locks, not every lot that is already scheduled is of interest. Most lots that have been scheduled before have already finished processing and are therefore no longer of interest. We only need to look at the lots that are serviced in the cluster tool at the same time, because only those lots will have an effect on each others cycle time (Figure 4).

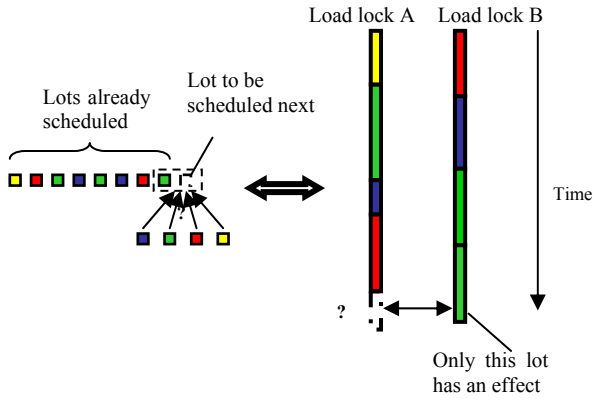

Figure 4: Evaluation Criteria

We will now describe three evaluation functions that do exactly the comparison of lot combinations as described above. The first two have already been presented in (Oechsner and Rose 2005), but still a quick overview is given in 3.3.1 for sake of completeness.

3.3.1 Slowdown Factor Comparison

As described in the last section, this evaluation function concentrates only on the last lot that was already scheduled and tries to find lots not yet scheduled that work well in combination with it. To this effect, the slowdown factor of all possible two-lot-combinations is computed, with the slowdown factor being defined as follows (See also (Niedermayer and Rose 2003)).

$$slowdown(A,B) = \frac{CycleTime(A \mid A \cup B)}{CycleTime(A)}$$

where CycleTime(A|AUB) is the mean cycle time of a wafer of lot A when it is serviced at the same time as lot B. CycleTime(A) is the mean cycle time of a wafer of lot A with only this lot being processed in the cluster tool. Since this slowdown factor is much higher if two lots do not fit together well, e.g., when they compete for a resource, it is a good indicator for evaluating lot combinations. All these cycle times are obtained by short simulations and the computed slowdown factors are stored for easy access by the filtered beam search algorithm.

We can expand this method by accounting for a special case. We assume that at least two lots have been scheduled already, and call the last two lots 'i-1' and 'i' respectively (according to their position in the schedule). Lot i is the last lot scheduled, but has a small cycle time compared to the cycle time of lot i-1. This means that lot i-1 is still being processed when lot i has left the cluster tool. Of course, the lot to be scheduled next must be compatible with lot i-1, not lot i.

In this case, we try to find a lot that has a low slowdown factor if processed at the same time as lot i-1. To decide which method to use, we look at the cycle times of lot i-1 and lot i. If CycleTime(i - 1) ≥ 4 * CycleTime(i), there is a high probability that lot i will be finished before lot i-1, and we choose the method described. If the condition above is not true, we use the normal method that computes the slowdown factors for the last lot scheduled. The factor of 4 is chosen based on estimation and may be tweaked.

3.3.2 Recipe Comparison

With this evaluation function, we concentrate on the last lot that is already scheduled and try to find lots that don't compete for too many resources with it.

A lot set consists of subsets of lot types. A lot type is defined by the recipe of the wafers of this lot as well as the number of wafers in it (all wafers in one lot have the same recipe). A recipe is the exact sequence and duration of the processing steps a wafer must be subjected to in order to manufacture the final product.

In our case, this simply means a list of chambers with associated process times for that chamber. However, because there is possibly more than one chamber for a certain process step, alternatives can be noted for each step, thus allowing a more flexible production.

Since we analyze only one cluster tool, all recipes we use feature chambers of that cluster tool only.

The recipes we consider have the following basic structure:

```
recipe := <step> [<step>]*
step := <chamber><time> [<chamber><time>]*
chamber := String
time := int
```

An example taken from a CluSim (a cluster tool simulator, see (Schmidt 1999)) definition file therefore looks like this:

```
Recipe RecipeA
    Step Step1
        ClusterTool ETCH1
            Chamber ChamberA 587
            Chamber ChamberB 585
    Step Step2
        ClusterTool ETCH1
            Chamber ChamberC 583
            Chamber ChamberD 586
```

In this example, chamber A or chamber B can be used for the first step, and the second step needs chamber C or chamber D. The integer numbers denote how long a wafer must be in the according chamber in order to finish the step. The time unit is seconds.

Since the information which resources are used by a lot can be found in its recipe, we simply compare the recipes of the two lots we consider. If they both use the same resource anywhere in their recipes, the resulting value used for computing the evaluation is higher, if alternatives exist, the penalty is lowered. The penalty formula is

$$p = \frac{1}{\#Alt_A \cdot \#Alt_B}.$$

$\#Alt_A$ is the number of alternatives in the step of recipe A where the coincidence was found, and $\#Alt_B$ is the same for recipe B. The total evaluation is the sum of all penalties, i.e., the sum over all coincidences.

We will illustrate this with an example. Below are two simple recipes in the format used by our simulator.

```
Recipe Recipe1
    Step Step1
        ClusterTool ETCH1
            Chamber ChamberA 534
            Chamber ChamberB 487
    Step Step2
        ClusterTool ETCH1
            Chamber ChamberC 395

Recipe Recipe2
    Step Step1
        ClusterTool ETCH1
            Chamber ChamberA 565
    Step Step2
        ClusterTool ETCH1
            Chamber ChamberD 349
```

Both recipes use chamber A. Recipe 1 could also use chamber B in that step, while recipe 2 has to use chamber A. Then the penalty that is added to the total would be

$$\frac{1}{2 \cdot 1} = \frac{1}{2}.$$

However, if there were also one alternative in recipe 2 (like in recipe 2b), the penalty would be

$$\frac{1}{2 \cdot 2} = \frac{1}{4}.$$

```
Recipe Recipe2b
    Step Step1
        ClusterTool ETCH1
            Chamber ChamberA 565
            Chamber ChamberE 756
    Step Step2
        ClusterTool ETCH1
            Chamber ChamberD 349
```

Since the computation of these values can be done when the recipes are known, the actual evaluation done during the beam search is just a lookup of the pre-computed value. This means that the recipe comparison is at least as fast as the methods using slowdown factors, which also do the actual work once before the search itself.

The recipe comparison method, while being simple, has a distinct disadvantage: it does not consider processing times of lots in the chambers of the cluster tool. Whether a lot has a processing time of 100 seconds or 100,000 seconds makes no difference to the evaluation function, since only the appearance (or, to be more exact, the number of appearances) of the chamber in the recipe counts. While that is no big drawback when processing times are roughly equal in all recipes, it could produce wrong results if this is not the case.

4 SIMULATION RESULTS

In this section, we show the results taken from simulations done with the CluSim cluster tool simulation software (Schmidt 1999). We added the filtered beam search algorithm and the evaluation methods described above to the simulator, and then conducted simulation runs for different lot set sizes.

We compare our evaluation methods for lot sets of the size 5, 10, 15 and 20. We chose a simplified beam search algorithm for each of the methods described, i.e., the filter step is discarded and only one evaluation is performed. We can do this because none of our methods is very time-consuming. The beam width parameter is set to the lot set size.

We simulated 22 randomly generated scenarios (i.e., lot recipes and distributions) and deleted the best and the

worst result. Thus, we have 20 scenarios for each parameter setting. Our primary performance criterion is the distance to the optimum/best found value obtained via a exhaustive/random search. In case of a small number of permutations, exhaustive search is used. However, the number of possible lot combinations gets huge quickly, so for the larger lot sets, we had to use a random search that covers only a part of the solution space. The amount of sequences tested is 10% of the total number of sequences with this method, but it is capped at 15000 runs due to time constraints.

However, it is also of interest whether the results are below or above the mean cycle time of all tested lot sequences. Since the simulations themselves are deterministic, there is no need to simulate more than once per scenario.

4.1 Slowdown Factors with Two Considered Lots

Table 1 shows that the slowdown factor comparison algorithm performs well. It keeps the makespan below the mean value of all tested lot sequences. However, while the scheduler produces results close to the optimum for small lot sets, it's performing worse when more lots are to be considered (Figures 5 and 6). Still, it stays well below the mean value in most cases. While the algorithm performs very well on small lot sets, and produces good results for more lots, the cycle times it yields are well above 110% of the optimal cycle time, but still below 120%.

Table 1: Results for the Simple Slowdown Factor Comparison

Lot set size	Result of FBS in % of optimum (slowdown with 2 lots)	Result of FBS in % of medium value (slowdown with 2 lots)
5	102.32	91.76
10	111.23	94.71
15	112.18	96.47
20	116.85	99.38

We have to consider that the values for the minimum and maximum makespans are obtained by a random search in the permutation space for the set sizes 10, 15 and 20, so they are not necessarily the best and worst cases. However, we can also compute the sum of all lot cycle times in single mode, which was always above the longest found makespan in our scenarios. The worst case found by our search methods is close to this sum in all of our tests. This shows us that our random search is not far off the mark and our results cover a large range of the possible schedules and their results.

The following figures (Figures 5 and 6) show the performance of the algorithm for the individual scenarios. They depict the cycle times of the tested lot sequences over the numbers of the tested scenarios. The thick horizontal line marks the mean cycle time of all tested lot sequences, with the thin vertical line representing the range of results we get for those sequences. The dot is the best result achieved by the beam search algorithm. The triangles mark the sum of the lot cycle times.

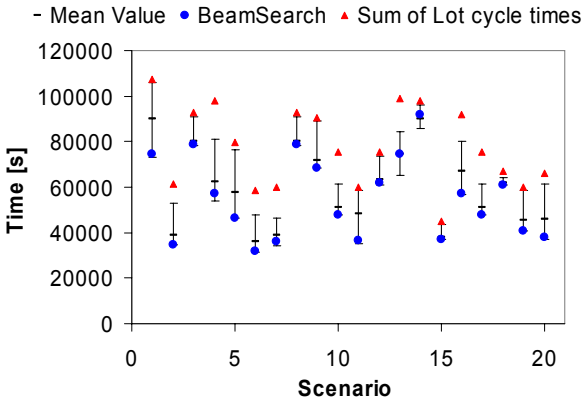

Figure 5: Lot Set Size of 5 Lots

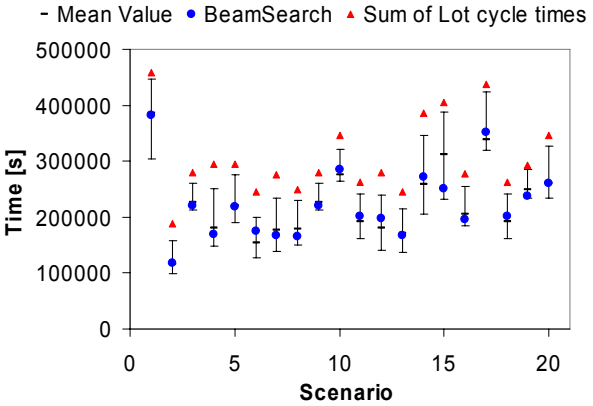

Figure 6: Lot Set Size of 20 Lots

4.2 Slowdown Factors with Three Considered Lots

A first look at Table 2 tells us that the expansion we made to the evaluation function with slowdown factors has an effect, if only a small one. In the case of a lot set size of 15, the improved method even is marginally worse, but still in the same range. However, a t-Test for corresponding pairs of scenarios with confidence level 0.95 showed that there are no statistically significant differences.

Table 2: Results for Methods Using Slowdown Factors

Lot set size	Result of filtered beam search in % of optimum		Result of filtered beam search in % of medium value	
	slowdown with 2 lots	slowdown with 3 lots	slowdown with 2 lots	slowdown with 3 lots
5	102.32	101.75	91.76	90.98
10	111.23	110.21	94.71	93.56
15	112.18	112.38	96.47	96.69
20	116.85	115.29	99.38	98.01

A closer look at the scenarios where we used the evaluation method with three considered lots shows that these setups contained recipes where the cycle times varied greatly. Between one third and half of the scenarios for each lot set size had at least one recipe pair with a cycle time quotient larger than 4. This is of course exactly the situation our improvement was aiming at. Still, with larger lot sets, the results are often in the range of the medium cycle time, with a big distance from the maximum (Figures 7 and 8).

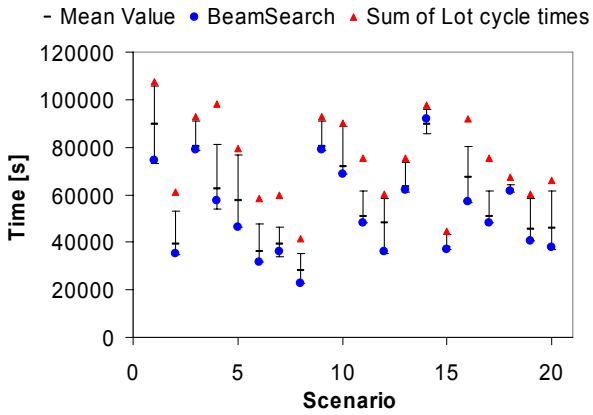

Figure 7: Lot Set Size of 5 Lots

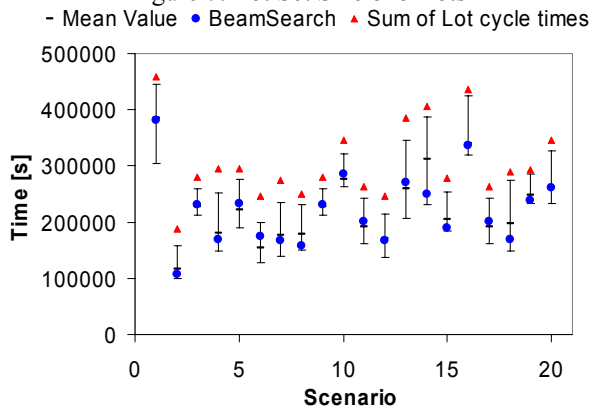

Figure 8: Lot Set Size of 20 Lots

4.2 Recipe Comparison

Surprisingly, the recipe comparison produces good results although it is the simplest method implemented. Although the comparison does not consider process times and although the process times given in the tested scenarios varied greatly, this evaluation method seems to employ the right criteria. The cycle times achieved by the algorithm using recipe comparison are nearly always within 110% of the best value found by the exhaustive or random search. Table 3 gives the mean results for the different lot sets in percent of the best found value and of the found mean value.

Table 3: Results for the Recipe Comparison

Lot set size	Result of FBS in % of optimum	Result of FBS in % of medium value
5	100.24	90.4
10	104.68	89.57
15	106.68	92.47
20	109.64	93.06

Also, the results are 10% lower than the medium value in the best case, and still show an improvement of 7% in the worst case. It has also to be considered that the reference values were computed with a much higher effort than the filtered beam search took. Figures 9 and 10 show how well the algorithm performed with the different scenarios for small and large lot sets.

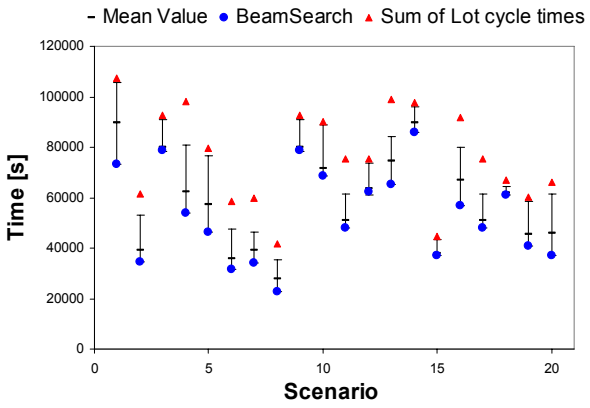

Figure 9: Lot Set Size of 5 Lots

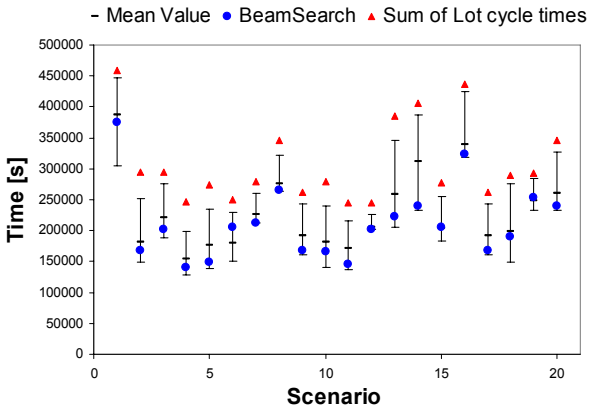

Figure 10: Lot Set Size of 20 Lots

Table 4: Result comparison for the different methods. All values are in % of optimum

Lot set size	slowdown with 2 lots	slowdown with 3 lots	Recipe comparison
5	102.32	101.75	100.24
10	111.23	110.21	104.68
15	112.18	112.38	106.68
20	116.85	115.29	109.64

The values in Tables 4 and 5 show that the recipe comparison performs better than the slowdown factor methods. Especially for larger lot sets, the results achieved with the newer method are significantly better.

Table 5: Result Comparison for the Different Methods. All Values are in % of Mean Value

Lot set size	slowdown with 2 lots	slowdown with 3 lots	Recipe comparison
5	91.76	90.98	90.4
10	94.71	93.56	89.57
15	96.47	96.69	92.47
20	99.38	98.01	93.06

4.3 Parameter Study

We will now take a closer look at how the beam width parameter influences the result of the algorithm. Since we have found that the recipe comparison method performs better than the slowdown factor approach, we will use this evaluation method from now on.

Basically, there is a trade off between the quality of the result and the speed of the search. The larger the beam width is, the better we can expect our solution to be. However, a large beam width also means that more simulation runs have to be conducted at the end of the search phase.

Since the time needed for the simulations takes by far the biggest part of the search algorithms running time, more simulations mean a significantly longer search time (The pure beam search itself without simulations takes about the same time as two simulations with a lot set size and also a beam width of 20.).

Figure 11 shows the influence of the beam width on the quality of the result. There is a clear trend to better results with a larger beam width, as was expected. However, there is no further advantage to expanding the beam width above a certain limit, which seems to be located at the lot set size for 10 lots.

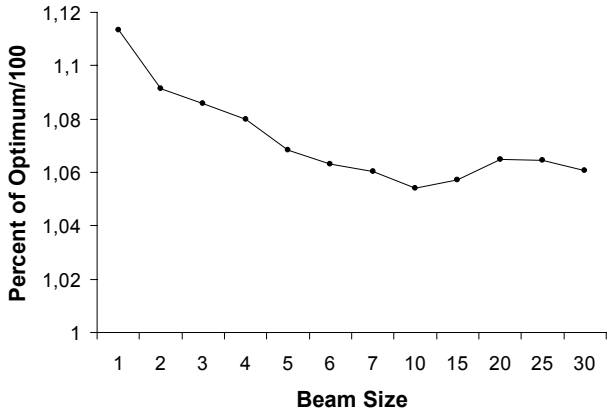

Figure 11: Results for 10 Lots

Surprisingly, a beam width of ten seems to be optimal for larger lot sets, too. Although it should be expected that beam widths grow with the lot set sizes, our parameter study shows that this is not the case. Figure 12 shows that the quality of the result even decreases with larger beam widths.

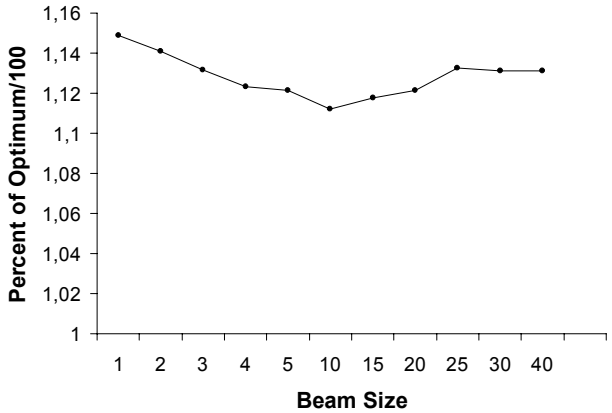

Figure 12: Results for 20 Lots

While the difference between the values decreases for a lot set size of 30 (Fig. 13), a beam width between 10 and 20 seems to be sufficient for these scenarios, too. However, because the performance of the algorithm generally becomes worse with larger lot sets, the difference between the parameter settings is not as significant as for example for a lot set size of 10.

A possible explanation for the observed behavior is the nature of our evaluation functions. Because of the fact that we only try to optimize the schedule locally, i.e., only pairs of lots are considered, it is conceivable that a larger beam width leaves more room for error. A wider beam means that there are more possible lot combinations to consider in the next step. While some of those combinations might seem good at that time, they could produce a schedule with a longer cycle time than others. However, because the algorithm is not able to look ahead, the best combination that is available at the moment is still chosen.

5 CONCLUSIONS

In this work, we studied a filtered beam search approach to scheduling cluster tools. The aim of the algorithm is to optimize the makespan for a given lot set by finding a good external schedule. While we already had promising evaluation functions that used slowdown factors, we searched for a different method that still uses knowledge about the system.
We developed, implemented and tested a new evaluation method based on recipe comparison that enhances the performance already shown by earlier approaches without needing more resources. The achieved results were significantly better than the makespans obtained by the slowdown factor comparison methods.

We also conducted a parameter study that showed that we do not need to expand the number of simulations above a certain limit to gain good solutions.

In conclusion, we showed that the already promising results of a filtered beam search approach can be even made better by choosing better evaluation functions. However, it seems that exploiting knowledge about the inner workings of a cluster tool is a good way to determine lot schedules.

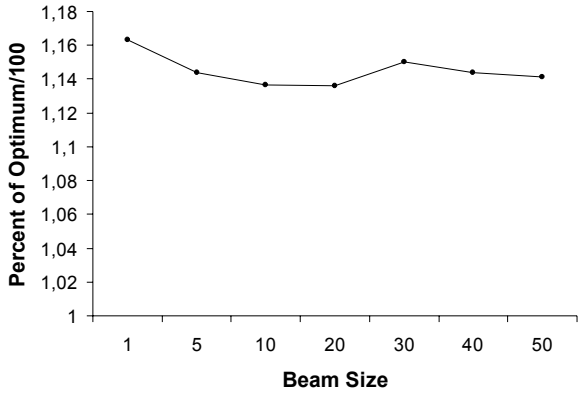

Figure 13: Results for 30 Lots

REFERENCES

Bohr, M. 1999. Schedulingverfahren für Cluster Tools in der Halbleiterfertigung. (In German) Master's thesis. Department of Computer Science. University of Würzburg, Germany.

Dümmler, M. 2004. Modeling and Optimization of Cluster Tools in Semiconductor Manufacturing. PhD thesis, Department of Computer Science. University of Würzburg, Germany.

Niedermayer, H., Rose, O.. 2003. A Simulation-based Analysis of the Cycle Time of Cluster Tools in Semiconductor Manufacturing. *Proceedings of 15th European Simulation Symposium,* Delft, Netherlands.

Oechsner, S. and Rose, O. 2005. A filtered beam search approach to scheduling cluster tools in semiconductor manufacturing. In *Proceedings of IERC'05*, May 14-18, Atlanta, USA

Pinedo, M. 2001. *Scheduling. Theory, Algorithms, and Systems*. 2nd edition. Prentice-Hall.

Schmidt, M. 1999. Modellierung und Simulation von Cluster Tools in der Halbleiterfertigung. (In German) Master's thesis. Department of Computer Science. University of Würzburg, Germany.

AUTHOR BIOGRAPHIES

SIMON OECHSNER is a PhD student and member of the scientific staff of Prof. Dr.-Ing. P. Tran-Gia at the Department of Distributed Systems at the University of Würzburg, Germany. He received a M.S. degree in computer science from Würzburg University, Germany. His research interests include the performance evaluation of peer-to-peer architectures and the simulation of distributed systems, such as manufacturing facilities. His Web address is http://www3.informatik.uni-wuerzburg.de/staff/oechsner/.

OLIVER ROSE holds the Chair for Modeling and Simulation at the Institute of Applied Computer Science of the Dresden University of Technology, Germany. He received an M.S. degree in applied mathematics and a Ph.D. degree in computer science from Würzburg University, Germany. His research focuses on the operational modeling, analysis and material flow control of complex manufacturing facilities, in particular, semiconductor factories. He is a member of IEEE, INFORMS Simulation Society, ASIM, and GI. His web address is www.iai.inf.tu-dresden.de/ms.